# LOG4J - QUICK GUIDE

Log4j is a Reliable, Fast and Flexible Logging Framework (APIs) written in Java which is distributed under the Apache Software License.

Log4j has been ported to the C, C++, C#, Perl, Python, Ruby, and Eiffel languages.

Log4j is highly configurable through external configuration files at runtime. It views the logging process in terms of levels of priorities and offers mechanisms to direct logging information to a great variety of destinations, such as a database, file, console, UNIX Syslog etc.

Log4j has three main components:

- **loggers:** Responsible for capturing logging information.
- **appenders :** Responsible for publishing logging information to various preferred destinations.
- **layouts:** Responsible to format logging information in different styles.

## log4j Features:

- log4j is thread-safe.
- log4j is optimized for speed.
- log4j is based on a named logger hierarchy.
- log4j supports multiple output appenders per logger.
- log4j supports internationalization.
- log4j is not restricted to a predefined set of facilities.
- Logging behavior can be set at runtime using a configuration file.
- log4j is designed to handle Java Exceptions from the start.
- log4j uses multiple levels, namely ALL, TRACE, DEBUG, INFO, WARN, ERROR and FATAL.
- The format of the log output can be easily changed by extending the *Layout* class.
- The target of the log output as well as the writing strategy can be altered by implementations of the Appender interface.
- log4j is fail-stop. However, altough it certainly strives to ensure delivery, log4j does not guarantee that each log statement will be delivered to its destination.

## log4j Installation:

Log4j API package is distributed under the Apache Software License, a fully-fledged open source license certified by the open source initiative.

The latest log4j version, including full-source code, class files and documentation can be found at http://logging.apache.org/log4j/.

Once downloaded apache-log4j-x.x.x.tar.gz, follow the given steps at log4j - Installation.

## log4j - Logging Levels:

The org.apache.log4j.Level class provides following levels but you can also define your custom levels by sub-classing the Level class.

| Level | Description |
|---|---|
| ALL | All levels including custom levels. |
| DEBUG | Designates fine-grained informational events that are most useful to debug an application. |
| ERROR | Designates error events that might still allow the application to continue running. |
| FATAL | Designates very severe error events that will presumably lead the application to abort. |
| INFO | Designates informational messages that highlight the progress of the application at coarse-grained level. |
| OFF | The highest possible rank and is intended to turn off logging. |
| TRACE | Designates finer-grained informational events than the DEBUG. |
| WARN | Designates potentially harmful situations. |

## Logging Methods:

Once we obtain an instance of a named logger, we can use several methods of the logger to log messages. The Logger class has the following methods for printing the logging information.

| SN | Methods with Description |
|---|---|
| 1 | **public void debug(Object message)**<br>This method prints messages with the level Level.DEBUG. |
| 2 | **public void error(Object message)**<br>This method prints messages with the level Level.ERROR. |
| 3 | **public void fatal(Object message);**<br>This method prints messages with the level Level.FATAL. |
| 4 | **public void info(Object message);**<br>This method prints messages with the level Level.INFO. |
| 5 | **public void warn(Object message);**<br>This method prints messages with the level Level.WARN. |
| 6 | **public void trace(Object message);**<br>This method prints messages with the level Level.TRACE. |

All the levels are defined in the org.apache.log4j.Level class and any of the above mentioned method can be called as follows:

```
import org.apache.log4j.Logger;
public class LogClass {
   private static org.apache.log4j.Logger log = Logger
                                    .getLogger(LogClass.class);
   public static void main(String[] args) {
      log.trace("Trace Message!");
      log.debug("Debug Message!");
      log.info("Info Message!");
      log.warn("Warn Message!");
      log.error("Error Message!");
      log.fatal("Fatal Message!");
   }
}
```

When you compile and run LogClass program it would generate following result:

```
Debug Message!
Info Message!
Warn Message!
Error Message!
Fatal Message!
```

## log4j - Log Formatting:

Apache log4j provides various Layout objects, each of which can format logging data according to various layouts. It is also possible to create a Layout object that formats logging data in an application-specific way.

All Layout objects receive a LoggingEvent object from the Appender objects. The Layout objects then retrieve the message argument from the LoggingEvent and apply the appropriate ObjectRenderer to obtain the String representation of the message.

## The Layout Types:

The top-level class in the hierarchy is the abstract class org.apache.log4j.Layout. This is the base class for all other Layout classes in the log4j API.

The Layout class is defined as abstract within an application, we never use this class directly; instead, we work with its subclasses which are as follows:

- DateLayout
- HTMLLayout ( Explained in this tutorial )
- PatternLayout. ( Explained in this tutorial )
- SimpleLayout
- XMLLayout

## The Layout Methods:

This class provides a skeleton implementation of all the common operations across all other Layout objects and declares two abstract methods.

| S.N. | Method & Description |
|---|---|

| | |
|---|---|
| 1 | **public abstract boolean ignoresThrowable()**<br>This method indicates whether the logging information handles any java.lang.Throwable object passed to it as a part of the logging event. If the Layout object handles the Throwable object, then the Layout object does not ignore it, and returns false. |
| 2 | **public abstract String format(LoggingEvent event)**<br>Individual layout subclasses will implement this method for layout specific formatting. |

Apart from these abstract methods, the Layout class provides concrete implementation for the methods listed below:

| S.N. | Method & Description |
|---|---|
| 1 | **public String getContentType()**<br>Returns the content type used by the Layout objects. The base class returns text/plain as the default content type. |
| 2 | **public String getFooter()**<br>Specifies the footer information of the logging message. |
| 3 | **public String getHeader()**<br>Specifies the header information of the logging message. |

Each subclass can return class-specific information by overriding the concrete implementation of these methods.

## log4j - Sample Program:

Following is a simple configuration file created for our example.It has the following information:

- The level of the root logger is defined as DEBUG and attaches appender named FILE to it.
- The appender FILE is defined as org.apache.log4j.FileAppender and writes to a file named "log.out" located in the **log** directory.
- The layout pattern defined is %m%n, which means the printed logging message will be followed by a newline character.

So the content of *log4j.properties* file would be as follows:

```
# Define the root logger with appender file
log = /usr/home/log4j
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

## Using log4j in Java Program:

The following Java class is a very simple example that initializes, and then uses, the Log4J logging library for Java applications.

```
import org.apache.log4j.Logger;

import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class log4jExample{
  /* Get actual class name to be printed on */
  static Logger log = Logger.getLogger(
                      log4jExample.class.getName());

  public static void main(String[] args)
                throws IOException,SQLException{

     log.debug("Hello this is an debug message");
     log.info("Hello this is an info message");
  }
}
```

## Compilation and Run:

Here are the steps to compile and run the above mentioned program. Make sure you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

All the libraries should be available in CLASSPATH and your *log4j.properties* file should be available in PATH. So do the following:

- Create log4j.properties as shown above.
- Create log4jExample.java as shown above and compile it.
- Execute log4jExample binary to run the program.

You would get following result, inside /usr/home/log4j/log.out file:

```
Hello this is an debug message
Hello this is an info message
```